Analisis Perbandingan Algoritma Backtracking Naif dan Backtracking Berbasis Heuristik Minimum Remaining Values pada Puzzle Shikaku

Safira Berlianti - 13524128
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: fira.berlianti@gmail.com , 13524128@std.stei.itb.ac.id

***Abstract*—Puzzle Shikaku merupakan puzzle logika berbasis grid yang dapat dimodelkan sebagai Constraint Satisfaction Problem (CSP). Algoritma backtracking umum digunakan untuk menyelesaikan CSP, tetapi pendekatan naif cenderung mengeksplorasi banyak cabang pencarian yang tidak menghasilkan solusi. Penelitian ini bertujuan membandingkan kinerja backtracking naif dan backtracking dengan heuristik Minimum Remaining Values (MRV) dalam menyelesaikan puzzle Shikaku. Kedua algoritma diimplementasikan menggunakan Python dan diuji pada 30 puzzle berukuran 5×5, 10×10, dan 15×15. Evaluasi dilakukan berdasarkan jumlah simpul pencarian, jumlah backtrack, dan waktu eksekusi. Hasil eksperimen menunjukkan bahwa MRV secara konsisten mengurangi jumlah simpul pencarian dan backtrack dibandingkan backtracking naif. Reduksi jumlah simpul mencapai 70,4% pada puzzle 5×5, 97,8% pada 10×10, dan 99,97% pada 15×15. Meskipun waktu eksekusi MRV sedikit lebih lambat pada puzzle kecil akibat overhead perhitungan kandidat, MRV memberikan percepatan yang signifikan pada puzzle berukuran sedang dan besar. Hasil penelitian menunjukkan bahwa heuristik MRV efektif meningkatkan efisiensi penyelesaian puzzle Shikaku, terutama untuk puzzle dengan ukuran yang lebih besar.**

Keywords—Shikaku, backtracking, Minimum Remaining Values (MRV), Constraint Satisfaction Problem (CSP)

I. PENDAHULUAN

Puzzle Shikaku merupakan puzzle berbasis grid yang mengharuskan pemain membagi area papan menjadi beberapa persegi panjang atau persegi sesuai angka yang diberikan. Setiap persegi panjang harus memuat tepat satu angka dan luas area persegi panjang tersebut harus sama dengan angka yang dimuatnya.

Penyelesaian puzzle Shikaku dapat dimodellkan sebagai *constraint satisfaction problem* (CSP), yaitu masalah penentuan nilai pada variabel-variabel tertentu sehingga seluruh kendala (*constraint*) yang diberikan dapat dipenuhi. Salah satu pendekatan yang umum digunakan untuk menyelesaikan CSP adalah algoritma *backtracking*. Pendekatan ini menjamin ditemukannya solusi apabila solusi memang ada, namun pada permasalahan dengan ruang pencarian yang besar, pendekatan *backtracking* naif sering kali menelusuri banyak cabang yang pada akhirnya tidak menghasilkan solusi. Untuk meningkatkan efisiensi pencarian, heuristik *Minimum Remaining Values* (MRV) dapat diterapkan. Heuristik ini bertujuan untuk mendeteksi kegagalan sedini mungkin sehingga ruang pencarian yang harus dieksplorasi dapat dikurangi.

Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk menganalisis perbedaan kinerja antara alioritma *backtracking* naif dan algoritma *backtracking* berbasis heuristik *Minimum Remaining Values* dalam menyelesaikan puzzle Shikaku. Analisis dilakukan melalui implementasi kedua pendekatan pada sejumlah puzzle dengan berbagai ukuran, kemudian dievaluasi menggunakan metrik berupa waktu eksekusi, jumlah simpul pencarian, dan jumlah proses *backtracking* yang terjadi selama pencarian solusi.

II. DASAR TEORI

A. Puzzle Shikaku

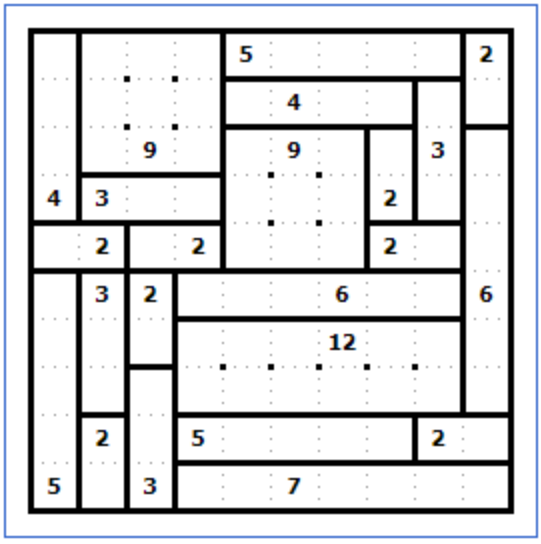

Fig. 1. Puzzle Shikaku. Sumber: https://www.puzzle-shikaku.com/

Shikaku merupakan puzzle logika berbasis grid yang bertujuan membagi seluruh sel pada papan menjadi beberapa daerah berbentuk persegi panjang atau persegi sesuai dengan angka yang diberikan pada beberapa sel tertentu. Setiap angka menyatakan luas daerah yang harus dibentuk, dan daerah tersebut harus memenuhi beberapa aturan, yaitu: (1) setiap daerah harus berbentuk persegi panjang atau persegi, (2) setiap daerah harus memuat tepat satu angka, (3) luas daerah harus sama dengan nilai angka yang dikandungnya, serta (4) seluruh papan harus tertutupi tanpa adanya tumpang tindih antar daerah.

Sebagai contoh, sebuah sel yang berisi angka 6 dapat membentuk daerah berukuran 1×6, 2×3, 3×2, atau 6×1 selama posisi daerah tersebut masih berada di dalam batas papan dan memenuhi seluruh *constraint* yang ada. Banyaknya kemungkinan pembentukan daerah inilah yang menyebabkan proses pencarian solusi menjadi semakin kompleks seiring bertambahnya ukuran puzzle.

B. Constraint Satisfaction Problem (CSP)

Constraint Satisfaction Problem (CSP) merupakan suatu permasalahan yang terdiri atas sekumpulan variabel, domain nilai yang mungkin untuk setiap variabel, serta sejumlah kendala yang harus dipenuhi secara bersamaan. Solusi dari sebuah CSP diperoleh dengan menentukan nilai untuk setiap variabel sedemikian rupa sehingga seluruh kendala yang ada tidak dilanggar.

Puzzle Shikaku dapat dimodelkan sebagai sebuah CSP. Pada penelitian ini, setiap sel yang berisi angka dipandang sebagai variabel yang perlu ditentukan persegi panjang penutupnya. Domain dari setiap variabel berupa seluruh kemungkinan persegi panjang yang dapat dibentuk dengan luas sesuai nilai angka pada sel tersebut. Selain itu, terdapat beberapa kendala yang harus dipenuhi, yaitu persegi panjang tidak boleh saling tumpang tindih, setiap persegi panjang harus memiliki luas yang sesuai dengan angka yang dikandungnya, serta seluruh sel pada papan harus tertutupi tepat satu kali.

C. Algoritma Backtracking

Algoritma *backtracking* adalah algoritma pencarian solusi berbasis *depth-first search* (DFS) yang membangun solusi secara bertahap dan melakukan *backtrack* apabila solusi parsial yang sedang dibentuk tidak dapat dikembangkan menjadi solusi lengkap. Dalam *backtracking*, seluruh kemungkinan solusi dari suatu persoalan dimodelkan dalam struktur pohon berakar yang disebut pohon ruang status.

Pencarian solusi dengan *backtracking* dilakukan dengan membangkitkan simpul-simpul status mengikuti aturan *depth-first order*. Simpul yang sedang diperluas disebut simpul-E (*expand node*), sedangkan simpul yang telah dibangkitkan dan masih mungkin menghasilkan solusi disebut simpul hidup (*live node*). Setiap kali simpul-E diperluas, lintasan yang dibangun bertambah panjang dengan menambahkan satu nilai untuk variabel berikutnya.

Agar pencarian efisien, *backtracking* menerapkan fungsi pembatas (*bounding function*) untuk menentukan apakah suatu simpul-E layak untuk diperluas. Jika fungsi pembatas menunjukkan bahwa lintasan yang sedang dibentuk tidak mungkin mengarah ke solusi (misalnya karena melanggar *constraint*), maka simpul-E tersebut dimatikan sehingga menjadi simpul mati (*dead node*). Proses pemangkasan cabang-cabang yang tidak menjanjikan ini disebut *pruning*. Jika pembentukan lintasan berakhir pada simpul mati, proses pencarian akan *backtrack* (kembali) ke simpul pada level di atasnya, kemudian dilanjutkan dengan membangkitkan simpul anak yang lain. Pencarian dihentikan ketika ditemukan simpul daun yang memenuhi seluruh *comstraint* (*goal node*).

Dalam implementasi *backtracking* naif untuk puzzle Shikaku, proses pencarian dilakukan dengan memilih sel bernomor berdasarkan urutan statis (misalnya mengikuti urutan membaca grid dari kiri ke kanan dan atas ke bawah). Untuk setiap sel yang dipilih, algoritma mencoba seluruh kandidat persegi panjang yang mungkin (dengan luas sesuai angka) secara berurutan. Fungsi pembatas yang digunakan adalah pemeriksaan apakah persegi panjang yang ditempatkan bertumpang tindih dengan persegi panjang lain atau keluar dari batas grid. Jika terjadi pelanggaran, simpul tersebut menjadi mati dan algoritma melakukan *backtracking* ke sel sebelumnya untuk mencoba kandidat persegi panjang lainnya. Pendekatan naif ini menjamin ditemukannya solusi apabila solusi memang ada. Namun, *backtracking* naif sering kali mengeksplorasi banyak cabang yang pada akhirnya tidak menghasilkan solusi. Hal ini menyebabkan jumlah simpul yang dikunjungi dalam pohon ruang status menjadi sangat besar, terutama untuk ukuran grid yang besar.

D. Minimum Remaining Values (MRV)

Minimum Remaining Values (MRV) adalah heuristik pemilihan variabel (*variable ordering heuristic*) yang umum digunakan dalam penyelesaian CSP. Prinsip dasar MRV adalah memilih variabel yang memiliki jumlah kandidat nilai paling sedikit (paling terbatas) untuk diproses terlebih dahulu. Heuristik ini merupakan penerapan dari prinsip *fail-first*, yaitu berusaha mendeteksi kemungkinan kegagalan sedini mungkin agar cabang-cabang pencarian yang tidak menjanjikan dapat dipangkas lebih awal. Berbeda dengan heuristik pemilihan nilai (*value ordering heuristic*), MRV tidak menentukan nilai mana yang harus dicoba terlebih dahulu, melainkan menentukan urutan variabel yang akan diproses selama proses pencarian.

Pada puzzle Shikaku, heuristik MRV diterapkan dengan memilih sel bernomor yang memiliki jumlah kandidat persegi panjang valid paling sedikit pada kondisi saat itu. Kandidat persegi panjang yang dimaksud adalah seluruh kemungkinan persegi panjang yang memiliki luas sesuai dengan angka pada sel, mencakup posisi angka tersebut, berada di dalam batas papan, serta tidak tumpang tindih dengan persegi panjang yang telah ditempatkan sebelumnya. Dengan memprioritaskan sel yang paling terbatas pilihannya, diharapkan konflik dapat teridentifikasi lebih awal sehingga jumlah simpul pencarian, jumlah proses *backtracking*, dan waktu eksekusi dapat berkurang dibandingkan pendekatan *backtracking* naif yang memproses sel berdasarkan urutan tetap.

Meskipun efektif dalam mengurangi jumlah simpul pencarian, penerapan MRV juga memiliki kelemahan berupa biaya komputasi tambahan (*overhead*) untuk menghitung

jumlah kandidat yang valid bagi setiap variabel pada setiap langkah pencarian. Pada puzzle dengan ukuran kecil atau jumlah variabel yang sedikit, *overhead* ini dapat menyebabkan waktu eksekusi MRV tidak berbeda signifikan atau bahkan lebih lambat dibandingkan *backtracking* naif.

III. METODE

A. Dataset Penelitian

Dataset yang digunakan dalam penelitian ini berupa kumpulan puzzle Shikaku yang diperoleh dari situs https://www.puzzle-shikaku.com/. Dataset yang dipilih terdiri dari 10 puzzle Shikaku untuk masing-masing ukuran grid 5×5, 10×10, dan 15×15. Dengan variasi ukuran ini, diharapkan dapat terlihat pada titik mana heuristik MRV mulai memberikan keuntungan yang signifikan dibandingkan pendekatan naif.

B. Implementasi Program

Program dikembangkan dalam bahasa pemrograman Python. Secara umum, program terdiri dari beberapa modul utama:

- Parser (parser.py): membaca file puzzle dan membangun representasi grid serta daftar petunjuk.
- Pembangkit kandidat (candidate_generator.py): membangkitkan seluruh kemungkinan persegi panjang yang valid untuk setiap petunjuk.
- Solver (naive_solver.py dan mrv_solver.py): mengimplementasikan algoritma backtracking naif dan backtracking dengan MRV.
- Metrik (metrics.py): mencatat jumlah simpul, jumlah backtrack, dan waktu eksekusi.

C. Representasi Puzzle

Setiap puzzle disimpan dalam file teks dengan format angka dipisahkan spasi. Simbol . atau 0 menyatakan sel kosong, sedangkan angka positif lainnya (2, 3, 4, dst.) menyatakan petunjuk (clue). Berikut adalah contoh file puzzle 5×5:

```
. . . 3 .
2 2 2 . .
. . . 4 .
. 2 . . 4
. 2 2 2 .
```

Dalam program, puzzle direpresentasikan sebagai:

- Grid: matriks dua dimensi (list[list[int]]) berisi bilangan bulat. Nilai 0 untuk sel kosong dan nilai >0 untuk petunjuk. Ukuran grid ditentukan dari jumlah baris dan kolom file input.
- Daftar clue: himpunan atau daftar tupel (baris, kolom, nilai) yang merepresentasikan posisi dan nilai setiap petunjuk. Daftar ini digunakan untuk iterasi selama proses pencarian solusi.

D. Pembangkitan Kandidat Persegi Panjang

Untuk setiap petunjuk, program membangkitkan seluruh kemungkinan persegi panjang yang dapat ditempatkan. Proses pembangkitan dilakukan dengan mencari semua pasangan (height, width) yang memenuhi height × width = value (luas sama dengan nilai petunjuk). Untuk setiap pasangan, program mencoba menempatkan persegi panjang di berbagai posisi yang masih mencakup posisi petunjuk tersebut.

Sebuah kandidat persegi panjang dinyatakan valid jika memenuhi syarat berikut:

- Luas area persegi panjang sama dengan nilai petunjuk (height × width = value).
- Persegi panjang berada di dalam batas grid (tidak keluar dari papan).
- Persegi panjang tidak tumpang tindih dengan persegi panjang lain yang telah ditempatkan sebelumnya.
- Persegi panjang hanya memuat tepat satu petunjuk, yaitu petunjuk yang sedang diproses. Persegi panjang tidak boleh memuat petunjuk lain di dalam areanya.

Proses pembangkitan kandidat ini dilakukan secara dinamis pada setiap langkah pencarian, karena validitas suatu kandidat bergantung pada konfigurasi persegi panjang yang sudah ditempatkan sebelumnya. Pendekatan dinamis ini memastikan bahwa keputusan yang diambil selalu berdasarkan keadaan terkini.

Berikut pseudocode untuk pembangkitan kandidat persegi panjang.

```
FUNCTION get_candidates(clue, state):
    KEMBALIKAN semua persegi panjang yang:
        1. luas = nilai clue
        2. berada dalam grid
        3. tidak tumpang tindih dengan state saat ini
        4. tidak memuat clue lain
```

E. Backtracking Naif

Backtracking naif mengimplementasikan strategi pencarian dengan memproses petunjuk berdasarkan urutan tetap sesuai dengan kemunculannya dalam daftar petunjuk (urutan membaca grid dari kiri ke kanan, atas ke bawah).

Pada setiap langkah, algoritma mengambil petunjuk berikutnya sesuai urutan tersebut, membangkitkan seluruh kandidat persegi panjang yang valid pada kondisi saat ini, lalu mencoba setiap kandidat satu per satu. Jika suatu kandidat berhasil ditempatkan tanpa melanggar kendala, algoritma melanjutkan ke petunjuk berikutnya secara rekursif. Jika tidak ada kandidat yang valid atau seluruh percobaan gagal, algoritma melakukan backtrack dengan membatalkan

keputusan terakhir dan mencoba kandidat lain pada tingkat sebelumnya.

Berikut pseudocode untuk backtracking naif.

```
FUNCTION naive_solve(clues, idx, state):
  IF idx == len(clues):
    RETURN semua sel tertutup

  clue ← clues[idx]
  FOR EACH rect IN get_candidates(clue, state):
    tempatkan rect ke state
    IF naive_solve(clues, idx+1, state):
      RETURN TRUE
    batalkan rect (backtrack)
  RETURN FALSE
```

F. Backtracking dengan Heuristik MRV

Berbeda dengan pendekatan naif, algoritma backtracking dengan MRV menerapkan strategi pemilihan petunjuk secara dinamis berdasarkan jumlah kandidat yang tersisa. Pada setiap langkah pencarian, algoritma melakukan langkah-langkah berikut:

- Untuk setiap petunjuk yang belum ditempatkan, hitung jumlah kandidat persegi panjang yang valid pada kondisi saat ini.
- Pilih petunjuk dengan jumlah kandidat paling sedikit (paling terbatas) untuk diproses terlebih dahulu.
- Jika terjadi tie (dua atau lebih petunjuk memiliki jumlah kandidat yang sama), petunjuk yang muncul lebih dulu dalam daftar dipilih (tanpa pemecah tambahan seperti degree heuristic agar fokus tetap pada MRV murni).

Strategi ini menerapkan prinsip fail-first dengan memilih petunjuk yang paling mungkin mengalami kegagalan terlebih dahulu, konflik dapat dideteksi sedini mungkin sehingga cabang-cabang pencarian yang tidak menjanjikan dapat dipangkas lebih awal. Hal ini diharapkan dapat mengurangi jumlah simpul yang dieksplorasi dan jumlah backtrack yang terjadi.

Untuk efisiensi, implementasi MRV dalam program ini menghindari pembangkitan kandidat dua kali. Pada saat menghitung jumlah kandidat untuk setiap petunjuk, program sekaligus menyimpan kandidat terbaik. Ketika petunjuk dengan domain terkecil terpilih, kandidat yang telah dibangkitkan langsung digunakan tanpa perlu membangkitkan ulang. Dengan demikian, overhead perhitungan MRV dapat ditekan seminimal mungkin.

Berikut pseudocode untuk backtracking dengan heuristik MRV.

```
FUNCTION mrv_solve(remaining_clues, state):
  IF remaining_clues KOSONG:
    RETURN semua sel tertutup

  // Pilih clue dengan jumlah kandidat paling sedikit
  best_clue ← NULL
  best_candidates ← NULL
  FOR EACH clue IN remaining_clues:
    cand ← get_candidates(clue, state)
    IF len(cand) < len(best_candidates):
      best_clue ← clue
      best_candidates ← cand

  remaining ← remaining_clues tanpa best_clue
  FOR EACH rect IN best_candidates:
    tempatkan rect ke state
    IF mrv_solve(remaining, state):
      RETURN TRUE
    batalkan rect (backtrack)
  RETURN FALSE
```

G. Metrik Evaluasi

Untuk mengukur kinerja, program mencatat tiga metrik selama proses pencarian berlangsung:

- Jumlah simpul (nodes): banyaknya state yang dieksplorasi selama pencarian. Setiap kali fungsi rekursif solve() dipanggil, counter ini bertambah satu.
- Jumlah backtrack (backtracks): banyaknya keputusan yang dibatalkan karena tidak dapat menghasilkan solusi yang valid. Counter ini bertambah setiap kali algoritma kembali ke tingkat sebelumnya setelah gagal menemukan kandidat yang layak.
- Waktu eksekusi (time_ms): waktu yang dibutuhkan untuk menemukan solusi, diukur dalam milidetik.

H. Skenario Eksperimen

Eksperimen dilakukan untuk membandingkan kinerja algoritma backtracking naif dan backtracking dengan heuristik Minimum Remaining Values (MRV) dalam menyelesaikan puzzle Shikaku. Kedua algoritma diimplementasikan menggunakan bahasa pemrograman Python dan dijalankan pada lingkungan pengujian yang sama untuk memastikan perbandingan yang adil.

Dataset yang digunakan terdiri atas puzzle Shikaku dengan tiga variasi ukuran grid, yaitu 5×5, 10×10, dan 15×15. Untuk setiap ukuran grid digunakan 10 puzzle yang diperoleh dari situs https://www.puzzle-shikaku.com/. Seluruh puzzle diproses menggunakan modul benchmark.py, sehingga kedua

solver menerima input yang identik dan dieksekusi dengan prosedur yang sama.

Pada setiap pengujian, mode verbose dinonaktifkan agar proses pencetakan langkah-langkah pencarian ke terminal tidak memengaruhi waktu eksekusi yang diukur. Setiap puzzle dijalankan sebanyak 3 kali untuk masing-masing solver, kemudian hasilnya dirata-rata untuk mengurangi noise dari sistem operasi. Hasil yang diperoleh kemudian dicatat secara otomatis ke dalam berkas keluaran untuk dianalisis lebih lanjut. Untuk menjaga validitas perbandingan, kedua solver menggunakan mekanisme pembangkitan kandidat (candidate generator) dan pemeriksaan kendala yang identik.

Seluruh eksperimen dijalankan pada komputer dengan spesifikasi: prosesor Intel Core i7-1165G7 @ 2.80 GHz, RAM 16 GB, sistem operasi Windows 11, dan Python 3.11. Hasil pengujian kemudian dievaluasi menggunakan metrik jumlah simpul pencarian (nodes), jumlah backtrack, dan waktu eksekusi.

IV. Hasil dan Pembahasan

A. Verifikasi Solusi

Untuk memastikan bahwa implementasi kedua solver telah bekerja dengan benar sesuai aturan Shikaku, dilakukan verifikasi pada salah satu puzzle dari dataset. Sebagai contoh, digunakan puzzle berukuran 5×5 dengan konfigurasi awal sebagai berikut.

```
. . 2 4 .
2 . 2 . .
2 . 3 . .
. 3 . 2 .
. 2 . 3 .
```

Fig. 2. *Contoh puzzle Shikaku 5×5 (input/5x5/5x5_06.txt)*

Hasil penyelesaian yang diperoleh dari naive solver ditunjukkan pada gambar berikut.

```
3  1  1  2  2
3  4  4  2  2
5  5  6  6  6
7  7  7  8  8
9  9 10 10 10
```

Fig. 3. *Solusi yang dihasilkan naïve solver*

Hasil penyelesaian yang diperoleh dari MRV solver ditunjukkan pada gambar berikut.

```
4  1  1  2  2
4  3  3  2  2
6  6  7  7  7
5  5  5  8  8
10 10  9  9  9
```

Fig. 4. *Solusi yang dihasilkan mrv solver*

Dari gambar di atas, meskipun penomoran persegi panjang berbeda, dapat diperiksa bahwa kedua solver memenuhi seluruh aturan puzzle Shikaku, yaitu setiap persegi panjang memiliki luas sesuai dengan nilai clue, tidak saling bertumpang tindih, menutupi seluruh grid, dan hanya mengandung satu clue. Hal ini menunjukkan bahwa kedua pendekatan mampu menghasilkan solusi yang benar.

B. Hasil Benchmark

Pengujian dilakukan terhadap 30 puzzle Shikaku yang terdiri atas tiga kelompok ukuran, yaitu 5×5, 10×10, dan 15×15. Setiap kelompok berisi 10 puzzle. Untuk memperoleh gambaran kinerja yang lebih stabil dan mengurangi noise dari lingkungan sistem, setiap puzzle dijalankan sebanyak 3 kali pada masing-masing solver, kemudian hasilnya dirata-rata.

TABLE I. Rata-rata benchmark

Ukuran	Solver	Solved	Waktu (ms)	Nodes	Backtracks
5×5	Naive	10/10	1,25	34,5	24,3
	MRV	10/10	1,64	10,2	0,0
10×10	Naive	10/10	143,53	1536,8	1514,8
	MRV	10/10	24,72	33,1	11,1
15×15	Naive	10/10	51339,66	326771,6	326732,5
	MRV	10/10	200,69	84,6	45,5

Untuk hasil lengkap setiap puzzle dapat diakses melalui repositori GitHub pada tautan berikut: https://github.com/Hwiyeahh/MakalahStima_13524128 (folder experiments/results.txt).

C. Pembahasan

Berdasarkan hasil benchmark pada Tabel 1, kedua solver berhasil menyelesaikan seluruh puzzle yang diuji pada ukuran 5×5, 10×10, maupun 15×15. Hal ini menunjukkan bahwa baik pendekatan backtracking naif maupun backtracking dengan heuristik Minimum Remaining Values (MRV) sama-sama mampu menghasilkan solusi yang valid sesuai aturan permainan Shikaku.

Meskipun tingkat keberhasilannya sama, perbedaan yang cukup besar terlihat dari sisi efisiensi proses pencarian. Solver MRV secara konsisten menghasilkan jumlah simpul pencarian (nodes) dan jumlah backtrack yang lebih sedikit dibandingkan solver naif. Fenomena ini dapat dijelaskan melalui prinsip fail-first yang menjadi dasar heuristik MRV. Pada setiap langkah

pencarian, MRV memilih clue yang memiliki jumlah kandidat persegi panjang paling sedikit untuk diproses terlebih dahulu. Dengan cara tersebut, kondisi yang tidak memungkinkan untuk menghasilkan solusi dapat terdeteksi lebih awal sehingga cabang-cabang pencarian yang tidak menjanjikan dapat segera dipangkas.

Efektivitas strategi tersebut terlihat dari penurunan jumlah simpul pencarian pada seluruh ukuran puzzle. Pada puzzle 5×5, rata-rata jumlah simpul berkurang dari 34,5 menjadi 10,2 simpul. Pada puzzle 10×10, jumlah simpul turun dari 1536,8 menjadi 33,1 simpul. Penurunan paling signifikan terjadi pada puzzle 15×15, yaitu dari rata-rata 326771,6 simpul menjadi hanya 84,6 simpul. Pola yang sama juga terlihat pada jumlah backtrack, yang menunjukkan bahwa MRV lebih efektif dalam menghindari eksplorasi terhadap cabang pencarian yang pada akhirnya tidak menghasilkan solusi.

Namun, pengurangan jumlah simpul tidak selalu berbanding lurus dengan penurunan waktu eksekusi. Pada kelompok puzzle 5×5, solver MRV justru memiliki waktu eksekusi rata-rata yang sedikit lebih tinggi dibandingkan solver naif. Hal ini menunjukkan adanya overhead komputasi akibat proses perhitungan ulang jumlah kandidat untuk setiap clue yang belum diproses. Pada ruang pencarian yang kecil, biaya tambahan tersebut belum dapat dikompensasi oleh manfaat pemangkasan cabang yang diperoleh dari heuristik MRV.

Sebaliknya, ketika ukuran puzzle bertambah, manfaat MRV menjadi semakin jelas. Pada puzzle 10×10, waktu eksekusi rata-rata turun secara signifikan dibandingkan solver naif. Perbedaan tersebut menjadi jauh lebih mencolok pada puzzle 15×15. Sebagai contoh, pada puzzle 15x15_06.txt, solver naif memerlukan lebih dari satu juta simpul pencarian dan backtrack sebelum menemukan solusi, sedangkan solver MRV hanya membutuhkan 109 simpul dan 73 backtrack. Temuan ini mengindikasikan bahwa pemilihan clue dengan domain paling kecil mampu mempersempit ruang pencarian secara drastis sehingga banyak keputusan yang tidak perlu dapat dihindari sejak tahap awal pencarian.

Hasil penelitian ini menunjukkan bahwa heuristik MRV efektif diterapkan pada permasalahan Constraint Satisfaction Problem (CSP) seperti Shikaku. Meskipun terdapat biaya tambahan pada proses pemilihan variabel, biaya tersebut relatif kecil dibandingkan penghematan yang diperoleh dari berkurangnya eksplorasi ruang pencarian, terutama pada puzzle berukuran sedang dan besar.

D. Analisis Kompleksitas

Misalkan ukuran grid adalah m×n, jumlah clue pada puzzle adalah N, dan rata-rata jumlah kandidat persegi panjang untuk setiap clue adalah K. Selain itu, misalkan D(v) menyatakan banyaknya faktor dari nilai clue v, A adalah luas maksimum suatu kandidat persegi panjang, dan C adalah jumlah clue pada grid.

1) Kompleksitas Pembangkitan Kandidat

Pertama, algoritma mencari seluruh pasangan (height, width) yang memenuhi height × width = value, dengan banyak kemungkinan sebanyak D(v). Untuk setiap pasangan tersebut, algoritma mencoba seluruh posisi persegi panjang yang mungkin pada grid sehingga tetap mencakup posisi clue. Selanjutnya, setiap kandidat yang terbentuk diperiksa validitasnya, yaitu dengan mengecek apakah terjadi overlap dengan persegi panjang yang sudah ditempatkan serta memastikan tidak terdapat clue lain di dalam area tersebut.

Berdasarkan tahapan tersebut, kompleksitas pembangkitan kandidat untuk satu clue dapat dinyatakan sebagai:

$$T_{candidate} = O(D(v) \times m\,n \times (A + C))$$

Dalam implementasi penelitian ini, nilai D(v) relatif kecil karena nilai clue pada puzzle Shikaku tidak terlalu besar. Selain itu, jumlah clue juga tidak terlalu banyak. Oleh karena itu, biaya pembangkitan kandidat umumnya tidak menjadi faktor dominan dibandingkan proses eksplorasi ruang pencarian. Meskipun demikian, secara teoretis kompleksitasnya tetap mengikuti persamaan tersebut.

2) Kompleksitas Backtracking Naif

Pada backtracking naif, urutan pemilihan clue bersifat tetap sesuai urutan yang diberikan pada input. Jika terdapat N clue dengan rata-rata K kandidat untuk setiap clue, maka pada kasus terburuk algoritma harus mengeksplorasi seluruh kemungkinan pada pohon pencarian.

Dengan demikian, kompleksitas waktunya dapat dinyatakan sebagai:

$$T_{naive} = O(K^N)$$

Kompleksitas eksponensial ini menjadi penyebab utama menurunnya kinerja pendekatan naif ketika ukuran puzzle semakin besar. Meskipun pemeriksaan validitas dapat memangkas sebagian cabang pencarian, pemangkasan tersebut tidak mengubah sifat eksponensial algoritma pada kasus terburuk.

3) Kompleksitas Backtracking dengan MRV

Secara teoretis, penggunaan heuristik MRV tidak mengubah kompleksitas kasus terburuk dari algoritma backtracking. Algoritma tetap berpotensi mengeksplorasi pohon pencarian yang bersifat eksponensial, sehingga batas atasnya tetap dapat dituliskan sebagai:

$$T_{MRV} = O(K^N)$$

Perbedaan utama terletak pada effective branching factor. Pada pendekatan naif, faktor percabangan rata-rata dapat dianggap sebesar K pada setiap level pencarian. Sebaliknya, MRV memilih clue dengan jumlah kandidat paling sedikit terlebih dahulu, sehingga faktor percabangan pada level-level awal cenderung lebih kecil.

Jika faktor percabangan efektif pada MRV dinyatakan sebagai $K_1, K_2, \ldots, K_N$ maka jumlah simpul yang dieksplorasi dapat diperkirakan sebagai:

$$Nodes_{MRV} \approx K_1 \times K_2 \times \cdots \times K_N$$

yang umumnya jauh lebih kecil dibandingkan K^N. Dengan kata lain, MRV tidak mengubah kompleksitas asimtotik, tetapi mampu mengurangi jumlah simpul pencarian yang benar-benar dieksplorasi melalui pemilihan variabel dengan domain terkecil.

Hal ini sejalan dengan hasil eksperimen pada penelitian ini. Pada puzzle berukuran 15×15, jumlah simpul pencarian rata-rata turun dari 326.771,6 simpul pada pendekatan naif menjadi hanya 84,6 simpul pada pendekatan MRV. Penurunan tersebut menunjukkan bahwa pemilihan clue yang lebih tepat dapat menghasilkan pemangkasan ruang pencarian yang sangat signifikan.

4) Implikasi terhadap Hasil Eksperimen

Pada puzzle berukuran 5×5, ruang pencarian efektif masih relatif kecil sehingga overhead akibat perhitungan ulang kandidat pada MRV menjadi lebih dominan. Perbedaan ini diperparah oleh frekuensi pemanggilan fungsi generate_candidates. Pada pendekatan naif, fungsi tersebut hanya dipanggil satu kali untuk setiap simpul pencarian, yaitu untuk clue yang sedang diproses. Sebaliknya, pada pendekatan MRV, fungsi tersebut dipanggil untuk setiap clue yang tersisa pada setiap simpul, karena algoritma harus menghitung jumlah kandidat semua clue untuk menentukan mana yang paling terbatas. Akibatnya, meskipun jumlah simpul pencarian lebih sedikit, waktu eksekusi MRV pada beberapa kasus justru sedikit lebih lambat dibandingkan pendekatan naif.

Sebaliknya, pada puzzle berukuran lebih besar, ruang pencarian meningkat secara drastis. Dalam kondisi ini, kemampuan MRV untuk memilih clue dengan domain terkecil mampu menurunkan faktor percabangan efektif dan mengurangi jumlah simpul yang harus dieksplorasi. Pengurangan jumlah simpul tersebut pada akhirnya mampu mengimbangi bahkan melampaui biaya tambahan yang diperlukan untuk menghitung ulang kandidat di setiap langkah.

V. Simpulan

Berdasarkan hasil implementasi dan eksperimen yang telah dilakukan, dapat disimpulkan bahwa algoritma backtracking naif maupun backtracking dengan heuristik Minimum Remaining Values (MRV) sama-sama mampu menyelesaikan seluruh puzzle Shikaku yang digunakan dalam penelitian ini. Kedua pendekatan terbukti menghasilkan solusi yang valid sesuai dengan aturan permainan Shikaku.

Berdasarkan hasil pengujian pada 30 puzzle dengan ukuran 5×5, 10×10, dan 15×15, heuristik MRV terbukti efektif dalam mengurangi jumlah simpul pencarian dan jumlah backtrack pada semua ukuran puzzle. Reduksi simpul mencapai 70,4% pada ukuran 5×5, 97,8% pada ukuran 10×10, dan 99,97% pada ukuran 15×15. Pada kasus paling ekstrem (puzzle 15×15_06), MRV berhasil menurunkan jumlah simpul dari lebih dari satu juta menjadi hanya 109 simpul. Secara teoretis, penurunan ini dapat dijelaskan melalui penurunan faktor percabangan efektif (effective branching factor) pada pohon pencarian.

Namun, efektivitas MRV dalam mengurangi ruang pencarian tidak selalu berbanding lurus dengan percepatan waktu eksekusi. Pada puzzle berukuran kecil (5×5), waktu eksekusi MRV justru sedikit lebih lambat dibandingkan backtracking naif karena keuntungan dari pemangkasan cabang tidak sebanding dengan besarnya overhead untuk menghitung kembali jumlah kandidat di setiap tahapan pencarian. Sebaliknya, pada puzzle berukuran sedang hingga besar (10×10 dan 15×15), pengurangan simpul yang sangat signifikan berhasil mengalahkan overhead tersebut, menghasilkan percepatan waktu eksekusi hingga 99,6% pada ukuran 15×15.

Dengan demikian, dapat disimpulkan bahwa heuristik MRV sangat direkomendasikan untuk penyelesaian puzzle Shikaku berukuran besar karena mampu memangkas ruang pencarian secara efektif melalui penerapan prinsip fail-first. Sementara itu, untuk puzzle berukuran kecil (seperti 5×5), pendekatan backtracking naif yang lebih sederhana sudah cukup efisien dan bahkan lebih cepat secara praktis karena tidak memiliki overhead perhitungan kandidat.

Gitbub Link

https://github.com/Hwiyeahh/MakalahStima_13524128

Video Link at Youtube

https://youtu.be/23pR9JuIHl8

Acknowledgment

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Allah SWT atas segala rahmat dan hidayah-Nya sehingga makalah penulisan ini dapat diselesaikan dengan baik. Apresiasi dan rasa terima kasih juga penulis sampaikan kepada para dosen kelompok keahlian Informatika serta para asisten kuliah IF2211 Strategi Algoritma, atas bimbingan, ilmu, dan arahan yang telah diberikan selama masa perkuliahan. Penullis juga sangat berterimakasih kepada orangtua penulis dan rekan-rekan penulis yang telah memberikan semangat, bantuan, dan dukungan kepada penulis.

References

[1] D. Vasubhai, "Shikaku Game Solved as CSP," Scribd, Jun. 2013. Diakses dari https://www.scribd.com/document/145570723/Shikaku

[2] Munir, R. (2026). Algoritma Runut-balik (backtracking) Bahan Kuliah IF2211 Strategi Algoritma (Bagian 1). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/15-Algoritma-backtracking-(2026)-Bagian1.pdf

[3] Munir, R. (2026). Algoritma Runut-balik (backtracking) Bahan Kuliah IF2211 Strategi Algoritma (Bagian 2). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/16-Algoritma-backtracking-(2026)-Bagian2.pdf

[4] Puzzle-Shikaku.com. (n.d.). *Shikaku Puzzles*. https://www.puzzle-shikaku.com/

[5] University of California, Berkeley. (n.d.). CS188: Introduction to Artificial Intelligence, Constraint Satisfaction Problems. Diakses dari https://www.cs.cmu.edu/~15281/coursenotes/constraints/index.html

[6] University of Cornell. (2011). CS4700: Artificial Intelligence, Lecture 5: Constraint Satisfaction Problems. Diakses dari https://www.cs.cornell.edu/courses/cs4700/2011fa/lectures/05_CSP.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2026

Safira Berlianti (13524128)